Multi-level Logic Synthesis Based on Pseudo-Kronecker Decision Diagrams and Local Transformation

Tsutomu Sasao, Hiraku Hamachi, Seiji Wada, and Munehiro Matsuura
Department of Computer Science and Electronics
Kyushu Institute of Technology
Iizuka 820, Japan

Abstract— This paper presents a method to derive multi-level logic networks using ANDs, ORs, EXORs and inverters. The design method first generates a pseudo-Kronecker decision diagram (PKDDs) from a given function. Then, it generates multi-level logic networks consisting of ANDs, ORs, EXORs and inverters. Finally, it simplifies the networks by using a local transformation method. Experimental results using MCNC benchmarks are shown.

Key words: multi-level logic synthesis, EXOR, Reed-Muller expansion, pseudo Kronecker decision diagram, BDD, local transformation.

I. Introduction

Most logic synthesis systems use design methods based on the AND, OR, and NOT operations. They produce excellent networks for control networks: In many cases, the qualitys of the produced networks are comparable to manually designed ones. However, for arithmetic networks, error correcting networks, and networks for telecommunications, manually designed ones are often much better than the ones produced by logic synthesis systems. To design such networks, we have to utilize EXOR gates in addition to AND, OR, and NOT gates. In this paper, we present a method to derive multi-level logic networks using ANDs, ORs, EXORs and inverters.

Various methods exist to generate multi-level networks. They can be classified in three ways according to the initial representation of the logic function:

1. Sum-of-products expressions (SOPs). For example, [8] uses algebraic factorization followed by local transformations to derive multi-level networks. MIS [1] is an enhanced system.

2. Exclusive-or sum-of-products expressions (ESOPs). For example, [19] uses algebraic factorization to derive multi-level networks, while [16] uses a graph coloring method to reduce the number of fan-in's in EXOR gates.

3. Decision diagrams (DDs). For example, [10] uses functional decision diagrams (FDDs), and [4] uses binary decision diagrams (BDDs) to derive multi-level networks, respectively.

If an ESOP representation of a function is simpler than the SOP representation, we should use the ESOP as the initial representation. However, for many functions, both SOPs and ESOPs are too large to generate. In such cases, we should use DDs, since they are often smaller than SOPs and ESOPs.

In a BDD, if each node is replaced by a two-input selector, we have a network for f. In this paper, we use PKDDs (pseudo Kronecker decision diagrams) to represent logic functions. PKDDs are generalization of BDDs, and require fewer nodes than BDDs [18]. We will show a method to derive multi-level networks consisting of ANDs, ORs, EXORs and inverters. First, the given function is represented by a PKDD, then each node of the PKDD is replaced by a module containing an EXOR gate (Fig. 1.1), and finally a multi-level logic network for f is realized. The network generated in this way contains many redundant gates. To remove such gates, we use a local transformation method [7]. This paper reports the design results for many benchmark functions. For arithmetic function, the synthesized networks require many fewer gates than ones generated by the conventional design methods.

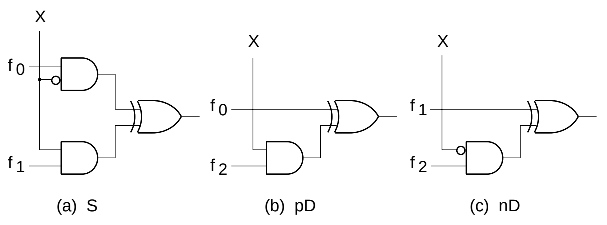


Figure 1.1: Circuits for the Shannon, the positive Davio, and the negative Davio expansions.

II. Binary Decision Diagrams

An arbitrary logic function f can be represented by

$$f = \bar{x}_1 f_0 \oplus x_1 f_1, \quad (2.1)$$
$$f = f_0 \oplus x_1 f_2, \text{ or} \quad (2.2)$$
$$f = \bar{x}_1 f_2 \oplus f_1, \quad (2.3)$$

where $f_2 = f_0 \oplus f_1$. (2.1) is the Shannon expansion, (2.2) is the positive Davio expansion, and (2.3) is the negative Davio expansion. For a given function f and x, the sub-functions f_0, f_1 and f_2 are unique. If we recursively apply the Shannon expansions to f_0 and f_1, then we have

$$f_0 = \bar{x}_2 f_{00} \oplus x_2 f_{01}, \text{ and}$$
$$f_1 = \bar{x}_2 f_{10} \oplus x_2 f_{11}.$$

In the similar way, if we apply the same expansion to f_{00} and f_{10}, we have

$$f_{00} = \bar{x}_3 f_{000} \oplus x_3 f_{001},$$
$$f_{01} = \bar{x}_3 f_{010} \oplus x_3 f_{011},$$
$$f_{10} = \bar{x}_3 f_{100} \oplus x_3 f_{101}, \text{ and}$$
$$f_{11} = \bar{x}_3 f_{110} \oplus x_3 f_{111}.$$

Fig.2.1 shows the Shannon tree. We can simplify this tree by using two rules:

1. If two sub-graphs represent the same functions, delete one, and connect the edge to the remaining sub-graph.

2. If both edges of a node point to the same sub-graph, delete that node, and directly connect its edge to the sub-graph.

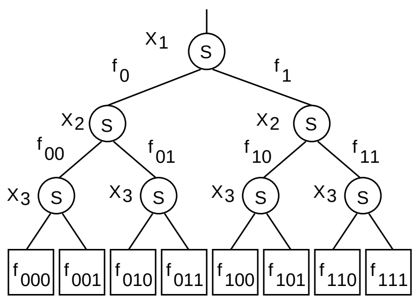


Figure 2.1: Shannon tree.

A reduced binary decision diagram (BDD) is obtained in this way. Given a function f and the order of the input variables, a unique BDD exists. Algorithms to manipulate BDDs have been developed, which are indispensable in logic synthesis [3].

III. PKDD

In the expansion tree for f, if any type of the three expansions is allowed for each node, then we have a decision tree shown in Fig.3.1. In this tree, the Shannon expansion is used for x_1, the positive and the negative Davio expansions are used for x_2, and all the three types of expansions are used for x_3. Such a tree is called a pseudo-Kronecker tree, and the simplified decision diagrams are called pseudo-Kronecker decision diagrams (PKDDs). For a function of n-variables, given an order of the input variables, there exist at most 3^{2^n-1} different PKDDs. A PKDD for f with the minimum number of nodes is called a minimum PKDD.

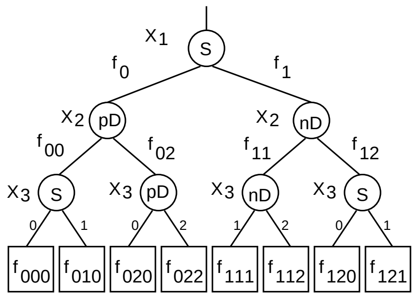


Figure 3.1: Pseudo-Kronecker tree.

IV. Optimization of PKDDs

In this part, we show a method to represent given functions by PKDDs using as few nodes as possible. In optimizating PKDD's, choices are made from two parameters:

1. Permutation of the input variables ($n!$ ways).

2. Selection of the expansion method for every node (3^{2^n-1} ways).

Note that there are $n!\ 3^{2^n-1}$ combinations. Even for small n, this is very large and it is virtually impossible to consider all these combinations when n is large.

To find a simple PKDDs efficiently, we use special a data structure called an EXOR ternary decision diagram (ETDD). Fig. 4.1 shows the EXOR ternary decision tree (ETDT) for a function $f(x_1, x_2, \ldots, x_n)$. Note that $f = \bar{x} f_0 \oplus x f_1$, where $f_0 = f(0, x_2, x_3, \ldots, x_n)$, $f_1 = f(1, x_2, x_3, \ldots, x_n)$ and $f_2 = f_0 \oplus f_1$. ETDTs for f_0, f_1, and f_2 are derived in similar ways.

An ETDD is a simplified version of an ETDT, and can be generated by using the following rules:

1. Merge sub-graphs representing the same functions.

2. Delete nodes representing $f = \bar{x} f_0 \oplus x f_1$ if $f_0 = f_1$.

Example 4.1 *Consider the function:*

$$f = \bar{x}_2 x_4 \oplus \bar{x}_1 x_2 \bar{x}_3 \oplus \bar{x}_1 x_2 x_4 \oplus x_1 \bar{x}_3.$$

To derive the ETDD for f, we obtain subfunctions:

$$f_0 = \bar{x}_2 x_4 \oplus x_2 \bar{x}_3 \oplus x_2 x_4,$$
$$f_1 = \bar{x}_2 x_4 \oplus \bar{x}_3,$$
$$f_2 = x_2 \bar{x}_3 \oplus \bar{x}_3 \oplus x_2 x_4 = \bar{x}_2 \bar{x}_3 \oplus x_2 x_4.$$

Next, we obtain sub-sub functions:

$$\begin{array}{lll} f_{00} = x_4, & f_{10} = x_3 \oplus \bar{x}_4, & f_{20} = \bar{x}_3, \\ f_{01} = \bar{x}_3 \oplus x_4, & f_{11} = \bar{x}_3, & f_{21} = x_4, \\ f_{02} = \bar{x}_3, & f_{12} = x_4, & f_{22} = \bar{x}_3 \oplus x_4. \end{array}$$

Fig. 4.2 shows the derivation tree for these sub-functions. Note that some sub-functions are the same. Thus, the ETDT in Fig. 4.2 is simplified into the ETDD in Fig. 4.3. In a similar way, we can derive the sub-functions, and finally, we obtain the ETDD shown in Fig. 4.4.

From here, we will derive a simple PKDD for the function to show the idea of the simplification algorithm. Fig. 4.3 shows that $f_{01} = f_{10} = f_{22}$ is more complex than other functions. Thus, to represent f_0, we use f_{00} and f_{02}; to represent f_1, we use f_{11} and f_{12}; and to represent f_2, we use f_{20} and f_{21}. In order to represent f, we arbitarily use f_0 and f_1, since the complexities of f_0, f_1, and f_2 are the same. Therefore, we have the PKDD shown in Fig. 4.5. (End of example.)

As shown in the previous example, an ETDD contains all the information necessary to derive a simple PKDD.

In a PKDD, if all the nodes are Shannon nodes, then it is a BDD. On the other hand, if all the nodes are positive Davio nodes, then it is a functional decision diagram (FDD) [10].

We use the following heuristic algorithm to find simple PKDDs.

Algorithm 4.1 *(A heuristic optimization of PKDDs)*

1. *Obtain orderings of the input variables that reduce the number of nodes in ETDDs, BDDs, and FDDs. Use simulated annealing to find good orderings of the input variables. For each of the three decision diagrams, perform the following steps, and choose the one with the fewest nodes.*
2. *Construct an ETDD.*
3. *As initial conditions for expansions, construct three types of decision diagrams, where*
   - *(a) All nodes represent the Shannon expansions,*
   - *(b) All nodes represent the positive Davio expansions, and*
   - *(c) All nodes represent the negative Davio expansions.*

   For each of three types of decision diagrams, perform step 4, and find the one with the fewest nodes.
4. *While we can reduce the number of nodes, do the followings:*
   From the root node down to the leaf nodes, change the method of expansions to each node. Count the nodes, when the expansions are S, pD, and nD, in order. Find the expansion that requires the fewest nodes, and adopt that expansion.

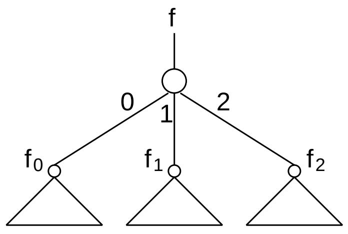


Figure 4.1: EXOR ternary decision tree.

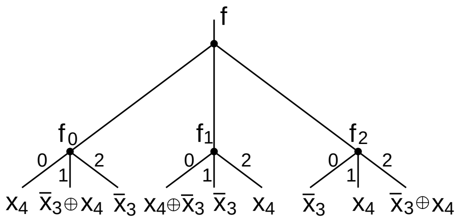


Figure 4.2: EXOR ternary decision tree for the function in Example 4.1.

Example 4.2 *Fig. 4.6 illustrates the optimization of expansion methods. At first, all the nodes are set to the Shannon nodes. Then, in the root node the best expansion is found from the three expansions. Suppose that the positive Davio expansion produces the best results in the root node. Then, the similar operations are done for each*

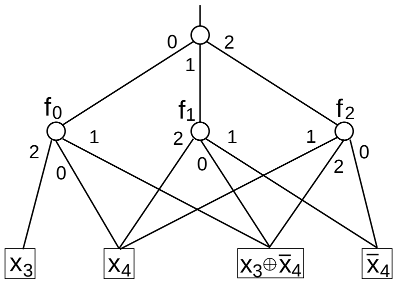


Figure 4.3: Derivation of EXOR ternary decision diagram in Example 4.1.

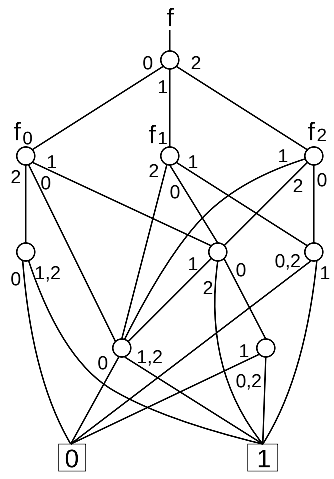


Figure 4.4: EXOR ternary decision diagram in Example 4.1.

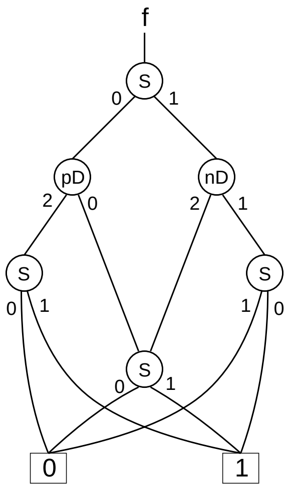


Figure 4.5: PKDD for the function in Example 4.1.

sub-tree. We continue this operation while we can reduce the number of nodes.

Fig. 4.7 shows the simplification of PKDD for $f = x_1 \oplus x_2$. Fig. 4.7(a) shows the EXOR TDD for f. If we use the positive Davio expansion in the root node, then we have Fig. 4.7(b). And, we can simplify the PKDD to obtain Fig. 4.7(c). Note that an ordinary BDD Fig. 4.7(d) requires three non-terminal nodes, while PKDD requires only two nodes.

V. Generation of Logic Networks

5.1. Strategy for expansions

In this paper, we assume that EXOR gates are more expensive than ANDs and ORs. When we expand a function $f = \bar{x}f_0 \vee xf_1$, we can choose the type of expansion

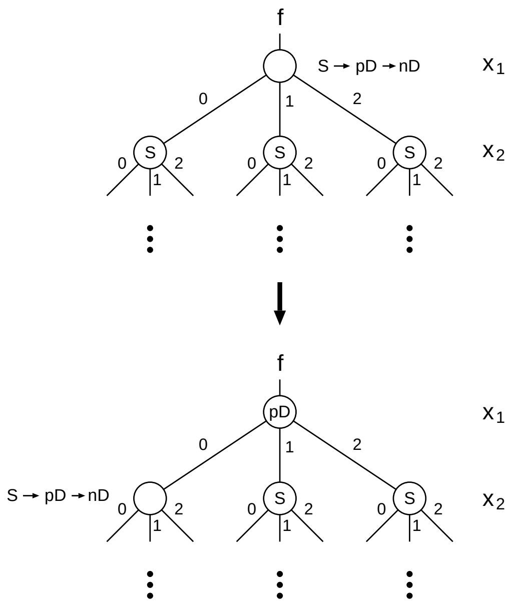


Figure 4.6: Selection of expansions.

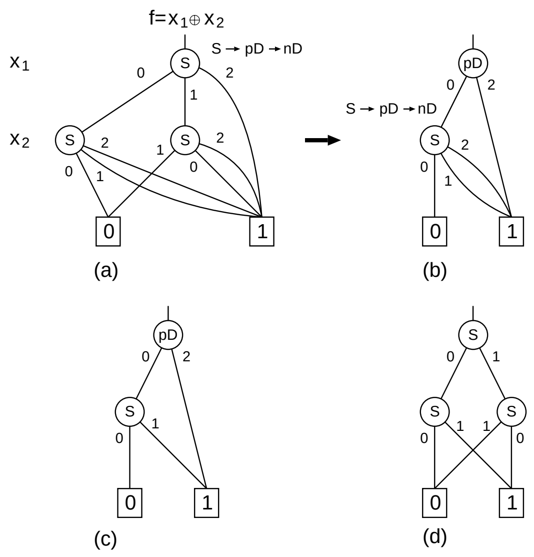


Figure 4.7: Simplification of decision diagrams for $x_1 \oplus x_2$.

by checking the implication relation of f_0 and f_1.

$f(x) \subseteq g(x)$, $(g(x) \supseteq f(x))$ denotes that $f(\boldsymbol{a}) \leq g(\boldsymbol{a})$. for each assignment of the input variables $\boldsymbol{a}$. $f \subset g$ denotes $f \subseteq g$ and $f \neq g$.

Theorem 5.1 *Let the given function be represented as $f = \bar{x}f_0 \oplus xf_1$.*

$$\textit{If} \quad f_0 \subseteq f_1 \quad \textit{then} \quad f = f_0 \vee xf_1, \tag{5.1}$$
$$\textit{if} \quad f_0 \supseteq f_1 \quad \textit{then} \quad f = \bar{x}f_0 \vee f_1. \tag{5.2}$$

Theorem 5.2 *Let the given function be represented as*

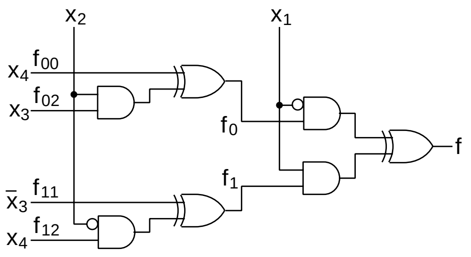


Figure 5.1: Network for the function in Example 4.1.

$f = g \oplus x^* h$.

$$\begin{aligned} &\textit{If}\ \ gh = 0 \quad \textit{then} \quad f = g \vee x^* h, && (5.3)\\ &\textit{if}\ \ g = 0 \quad \textit{then} \quad f = x^* h, \textit{and} && (5.4)\\ &\textit{if}\ \ g \supset h \quad \textit{then} \quad f = \overline{(x^* h)} \cdot g, && (5.5) \end{aligned}$$

where x^* denotes x or $\bar{x}$.

In the case of (5.1), f is positive with respect to x. In the case of (5.2), f is negative with respect to x. If (5.1) or (5.2) holds, then we use the Shannon expansion. When we generate multi-level logic networks, for the Shannon expansions we use the OR operators instead of the EXORs. The condition for (5.3) is essentially the same as (5.1) or (5.2). Also the condition for (5.4) is a special case of (5.3). In the case of (5.5), an EXOR gate is replaced by an AND gate and an inverter.

5.2. Generation of logic networks

In a PKDD, if each node is replaced by one of the circuits in Fig.1.1, we have a network for f. Usually, PKDDs require fewer nodes than BDDs. So, from PKDDs, we can expect networks with fewer gates than ones from BDDs.

Example 5.1 *Fig. 5.1 shows the network for the PKDD in Fig. 4.5. Note that if the both children of a node are constants, then the node can be removed, and a variable or its complement can be connected.*

(End of Example)

VI. Local Transformation

The following rules are used in the local transformations:

1. Simplification of ANDs and ORs (Fig. 6.1).
   Reduction of constants.
   Merging cascaded gates.
2. Inverter reduction (Fig. 6.2).
   Reduction of cascaded inverters.
   de Morgan's transformation.
3. Simplification of EXORs (Fig. 6.3).
   Reduction of constants.
   Merging cascaded gates.
   Extraction of inverters.
4. Simplification of AND-EXOR (OR-EXOR) (Fig. 6.4).
   Reduction of same inputs in the cascade.
5. Simplification considering the output function (Fig. 6.5).
   Detection of constant outputs.
   Sharing outputs.
   Sharing inverted outputs.
6. Simplification using implication relation (Fig. 6.6).
7. Simplification considering inputs and outputs (Fig. 6.7).

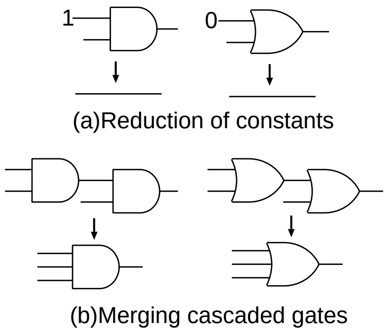
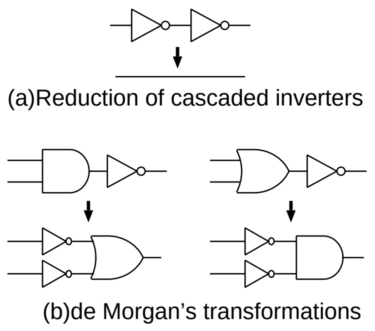


Figure 6.1: Simplification of ANDs and ORs.

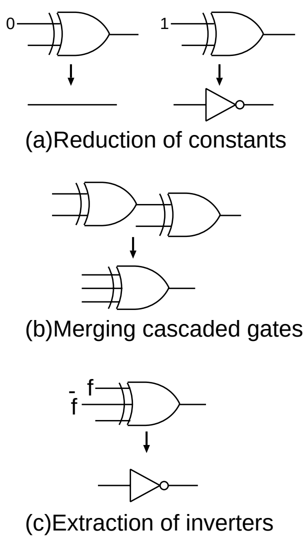


(a)Reduction of cascaded inverters
(b)de Morgan's transformations

Figure 6.2: Inverter reduction.

VII. Experimental Results

We developed programs to generate PKDDs from given functions, and to transform PKDDs into multi-level logic networks.



0
1
(a)Reduction of constants
(b)Merging cascaded gates
f
f̄
(c)Extraction of inverters

Figure 6.3: Simplification of EXORs.

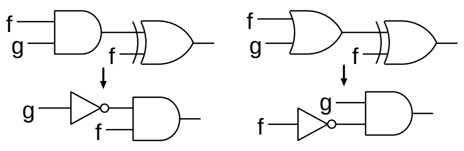


Figure 6.4: Simplification of AND-EXORs.

Tables 7.1 and 7.2 compare the numbers of non-terminal nodes in BDDs and PKDDs for various benchmark functions. In these tables, the columns headed "nodes" denote the number of nodes in the simplified decision diagrams.

These tables also compare the number of gates and interconnections. By using PKDDs, number of nodes, gates, and interconnections are reduced by, on the average, 21 percent.

Orderings of the input variables that minimize the number of the nodes in BDDs do not always minimize the number of nodes in PKDDs.

Fig. 7.1 shows the four-bit adder (ADR4) generated by the design method. It is a cascaded ripple carry adder. Up to $n = 6$, the algorithm produced the ripple carry adders.

For arithmetic networks, the generated networks require many fewer gates when we start from PKDDs instead of BDDs. For control networks, the generated networks require more gates than one generated by conventional two-level methods.

VIII. Conclusion and Comments

In this paper, we presented a method to generate multi-level logic network by using EXOR gates. This method

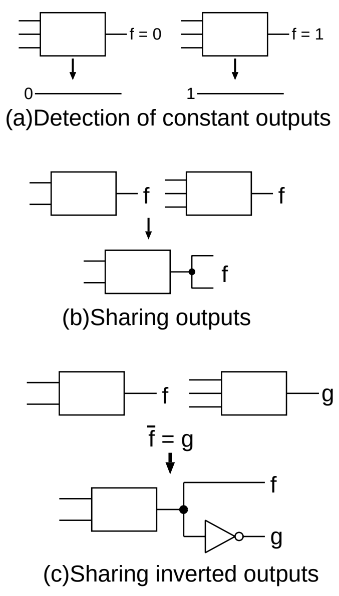


Figure 6.5: Simplification considering output function.

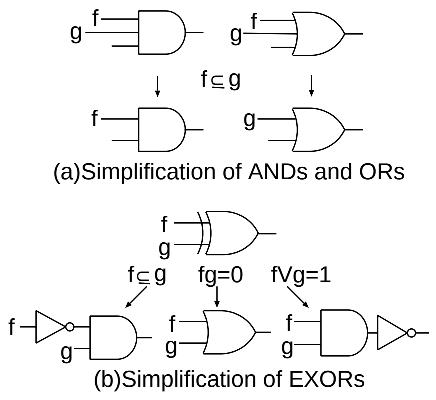


Figure 6.6: Simplification using implication relation.

first generates a PKDD for a given function, and then transforms it into a multi-level network, and finally sim-

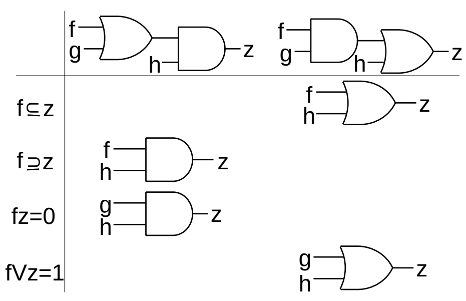


Figure 6.7: Simplification considering inputs and outputs.

Table 7.1: Optimization Results (BDD).

Function	In	Out	nodes	Gates			Fan-in		Levels
				NOT	AND	OR	AND	OR	
5xp1	7	10	68	21	33	30	68	61	10
add6	12	7	78	17	104	59	208	119	24
adr4	8	5	29	11	15	13	30	26	13
alu2	10	8	88	13	74	52	154	112	17
apla	10	12	102	10	81	51	185	103	16
bw	5	28	108	9	94	78	193	161	10
clip	9	5	105	18	97	58	196	129	17
co14	14	1	27	14	37	13	75	26	27
con1	7	2	64	6	12	10	25	21	7
dc2	8	7	64	7	53	35	116	70	13
dist	8	5	152	28	144	98	293	202	17
dk17	10	11	62	10	45	21	105	42	11
duke2	22	29	377	21	280	157	652	323	25
ex5	8	63	288	22	157	140	352	321	12
f51m	8	8	67	23	38	30	76	61	13
in2	19	10	232	17	269	163	565	334	22
in7	26	10	98	16	63	42	140	88	23
inc	7	9	72	16	58	39	127	79	13
misex1	8	7	38	10	29	20	67	40	9
misex3	14	14	585	50	505	319	1127	663	26
mlp4	8	8	141	12	160	111	330	224	17
radd	8	5	39	11	44	26	88	53	16
rd53	5	3	23	9	20	12	40	24	10
rd73	7	3	43	15	35	24	70	50	15
rd84	8	4	59	20	49	32	101	67	17
risc	8	31	67	8	49	21	107	45	8
rot8	8	5	75	15	57	42	121	86	15
sao2	10	4	85	10	77	43	161	100	17
sex	9	14	49	10	30	21	66	46	9
sqr8	8	16	233	33	253	171	518	343	16
sym6	5	2	15	5	11	8	22	18	7
sym9	9	1	33	9	31	20	62	44	14
t481	16	1	32	18	31	15	64	30	29
tial	14	8	692	82	609	377	1244	819	27
ts10	22	16	146	6	96	48	288	96	7
vg2	25	8	194	19	131	102	307	212	23
xor5	5	1	9	8	8	4	16	8	10
Total			4664	637	3891	2509	8390	5254	588

Table 7.2: Optimization Results (PKDD).

Function	In	Out	nodes	Gates				Fan-in			Levels
				NOT	AND	OR	EXOR	AND	OR	EXOR	
5xp1	7	10	33	12	17	5	14	35	10	30	12
add6	12	7	57	0	41	1	21	86	6	52	11
adr4	8	5	15	0	7	3	7	14	6	14	10
alu2	10	8	70	19	58	27	8	128	57	16	14
apla	10	12	96	10	81	51	0	185	103	0	16
bw	5	28	91	14	86	43	20	185	87	40	8
clip	9	5	79	19	62	39	12	127	87	29	15
co14	14	1	26	14	24	6	7	48	12	14	26
con1	7	2	15	6	13	9	0	27	19	0	7
dc2	8	7	56	8	39	21	5	90	44	11	12
dist	8	5	132	21	105	59	26	213	122	57	15
dk17	10	11	61	11	46	20	0	107	40	0	11
duke2	22	29	343	37	268	112	16	634	227	33	23
ex5	8	63	228	39	166	85	18	372	194	37	15
f51m	8	8	30	10	17	4	13	34	8	28	12
in2	19	10	205	23	253	149	2	536	301	4	23
in7	26	10	84	19	53	17	17	126	35	38	20
inc	7	9	65	12	53	27	11	115	55	23	11
misex1	8	7	34	9	29	18	1	66	36	2	9
misex3	14	14	500	36	437	222	90	953	452	185	26
mlp4	8	8	99	9	86	30	27	176	64	69	13
radd	8	5	26	0	16	1	10	34	4	23	7
rd53	5	3	13	0	8	2	6	16	4	13	7
rd73	7	3	21	0	14	2	12	28	6	25	9
rd84	8	4	29	0	18	2	16	39	6	33	10
risc	8	31	55	9	42	12	0	91	26	0	8
rot8	8	5	64	17	59	31	6	122	62	12	15
sao2	10	4	76	10	64	33	6	135	80	12	17
sex	9	14	44	14	34	11	0	81	23	0	9
sqr8	8	16	171	19	144	63	51	293	126	122	14
sym6	5	2	15	7	12	7	0	24	16	0	8
9sym	9	1	26	10	20	1	17	40	4	36	12
t481	16	1	17	10	9	0	3	22	0	9	6
tial	14	8	413	39	377	231	42	774	545	85	25
ts10	22	16	146	6	96	48	0	288	96	0	7
vg2	25	8	191	24	137	94	0	315	196	0	22
xor5	5	1	5	0	0	0	1	0	0	5	1
Total			3655	511	3003	1490	485	6590	3167	1057	492

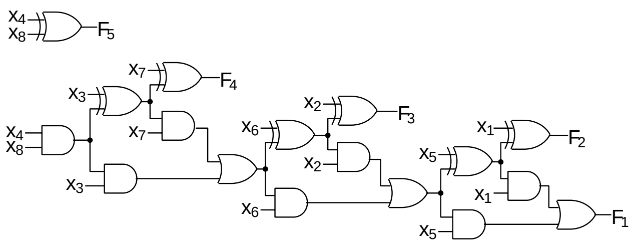


Figure 7.1: 4-bit adder generated by the algorithm.

plifies the network by local transformation methods.

Because PKDDs require fewer nodes than BDDs, they are promising for the initial solutions for the arithmetic networks. When, the given function is represented by a PKDD with a small number of nodes, this method is quite effective. For some functions, the conventional method based on AND-OR two-level expression produces simpler networks.

The presented method tends to produce networks with more levels than conventional ones, because of the followings reasons:

1. The present method mainly uses two-input gates as basic gates, while the conventional methods use gates with more inputs.

2. The present method assumes that only ANDs, ORs, EXORs, and inverters are basic elements. The number of gates and logical levels can be reduced by using NANDs, NORs, EXNORs etc.

In the practical design system, we have to decide which method to use. For the control networks, we should use the conventional method using AND-OR two-level logic. For the arithmetic networks, we should use the method presented. The problem is to find the best method to represent the given function.

Acknowledgments

This work was supported in part by a Grant in Aid for Scientific Research of the Ministry of Education, Science and Culture of Japan.

References

[1] R. K. Brayton, R. L. Rudell, A. Sangiovanni-Vincentelli, and A. R. Wong, "MIS: A multiple-level optimazation system," *IEEE Trans. CAD*, vol. CAD-6, No. 6, pp. 1062-1081, Nov. 1987.

[2] S. D. Brown, R. J. Francis, J. Rose, and Z. G. Vranesic, *Field Programmable Gate Arrays*, Kluwer Academic Publishers, Boston 1992.

[3] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. Comput.* Vol. C-35, No. 8, Aug. 1986, pp. 677-691.

[4] L. Burgun, N. Dictus, A. Greiner, E. Prado Lopez, and C. Sarwary, "Multilevel logic optimization of very high complexity circuits," *Proceedings of EURO-DAC '94*, 1994, pp.14-19.

[5] M. Davio, J-P Deschamps, and A. Thayse, *Discrete and Switching Functions*, McGraw-Hill International, 1978.

[6] R. Drechsler, A. Sarabi, M. Theobald, B. Becker and M. A. Perkowski, "Efficient representation and manipulation of switching functions based on ordered Kronecker Functional Decision Diagrams," Fachbereich Informatik, Universitat Frankfurt, Interner Bericht 14/93.

[7] K. Enomoto, S. Nakamura, T. Ogihara, and S. Murai, "LORES-2: A logic reorganization system," *IEEE Design and Test*, Oct. 1985, pp.35-42.

[8] D. Gregory, K. Bartlett, A. de Geus, and G. Hachtel, "Socrates: A system for automatically synthesizing and optimizing combinational logic," *Design Automation Conference 1986*, pp.79-85, June 1986.

[9] N. Ishiura, H. Sawada, and S. Yajima, "Minimization of binary decision diagrams based on exchange of variables," *ICCAD-91*, pp. 472-475, Nov. 1991.

[10] U. Kebschull, E. Schubert and W. Rosenstiel, "Multilevel logic synthesis based on functional decision diagrams," *EDAC 92*, 1992, pp. 43-47.

[11] U. Kebschull, E. Schubert, and W. Rosenstiel, "Multilevel logic synthesis based on functional decision diagrams," in *Proc. European Conference on Design Automation EDAC '92*, March 1992, pp. 43-47.

[12] S. Minato, N. Ishiura, and S. Yajima, "Shared binary decision diagram with attributed edges for efficient Boolean function manipulation," *27th Design Automation Conference*, pp. 52-57, June, 1990.

[13] L. McKenzie, L. Xu and A. Almaini, "Graphical representation of generalized Reed-Muller Expansions," *IFIP 10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design*, Hamburg, Sept. 16-17, 1993.

[14] T. Sasao, "Input variable assignment and output phase optimization of PLA's," *IEEE Trans. on Comput.* vol. C-33, No. 10, pp. 879-894, Oct. 1984.

[15] T. Sasao and P. Besslich, "On the complexity of MOD-2 Sum PLA's," *IEEE Trans. on Comput.* Vol.39. No.2, Feb. 1990.

[16] T. Sasao (ed.), *Logic Synthesis and Optimization*, Kluwer Academic Publishers, 1993.

[17] T. Sasao, "EXMIN2: A simplification algorithm for exclusive-OR Sum-of-products expressions for multiple-valued input two-valued output functions," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol.12, No.5, May 1993, pp.621-632.

[18] T. Sasao and J. T. Butler, "A design method for look-up table type FPGA by pseudo-Kronecker expansion," *Proc. International Symposium on Multiple-Valued Logic*, May 1994, pp.97-106.

[19] J. M. Saul, "Towards a mixed exclusive-/inclusive-or factored form," in *IFIP WG 10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design*, Sept. 1993, pp. 2-5.